NATIONAL UNIVERSITY OF SINGAPORE

CS1010 – PROGRAMMING METHODOLOGY

(Semester 1: AY2016/17)

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. This assessment paper consists of **FOURTEEN (14)** questions and comprises **THIRTEEN (13)** printed pages.
2. This is an **OPEN BOOK** assessment.
3. Calculators and electronic dictionaries are NOT allowed.
4. Answer all questions, and write your answers in the **ANSWER SHEETS** provided.
5. Fill in your Student Number with a pen, clearly on every odd-numbered page of your ANSWER SHEETS.
6. You may use **2B pencil** to write your programs.
7. Note that penalty will be given for programs that are unclear or unnecessarily long.
8. You must submit only the ANSWER SHEETS and no other document.

Section A: Multiple Choice Questions (MCQs) [12 marks]

There are six MCQs in this section. Each MCQ has one correct answer and is worth 2 marks. There is no penalty for a wrong answer.

Q1. Which of the following statements is **false** about random number generation?

A. **rand()** generates a random integer in [0, **RAND_MAX**].

B. The expression **rand()** % 5 * 2 + 3 generates a random odd integer in [3, 11].

C. **srand()** can be called more than once in a program.

D. The return value of **time(NULL)** can be used as the input for **srand()**.

E. As long as **srand()** is called once before **rand()** is called, the sequence of random numbers generated changes every time the program is executed.

Q2. How many of the following code fragments set all the elements in the array **arr** to 1?

(i)
```
int arr[2][3] = {{1,1},{1},{1,1,1}};
```

(ii)
```
int arr[2][3], i;
    for (i = 0; i < 2; i++)
        arr[i] = {1,1,1};
```

(iii)
```
int arr[2][3], arr2[3] = {1,1,1}, i;
   for (i = 0; i < 2; i++)
      arr[i] = arr2;
```

(iv)
```
int arr[2][3] = {{0}}, i, j;
  while (i < 2){
     j = 0;
     while (j < 3){
        arr[i][j]++;
        j++;
     }
     i++;
  }
```

A. 0

B. 1

C. 2

D. 3

E. 4

Q3. Which of the following statement is **true** at the end of this code fragment?

```
char str1[] = {'p', 'e', 'n', '\0', 'c', 'i', 'l', '\0'};
char str2[14];
strcpy(str2, "apple");
strcat(str2, str1);
```

A. **str2** contains the string "apple".
B. **str2** contains the string "applepen".
C. **str2** contains the string "applepencil".
D. **str2** contains the string "pen".
E. **str2** contains the string "pencil".

Q4. Consider the following structure type definition:

```
typedef struct {
    char name[50];
    char grade;
} student_t;
```

What is the output of the following code fragment?

```
int main(void){
    student_t stu[2] = {{"Tan",'F'}, {"Kim",'B'}};

    func1(stu);
    func2(&stu[1].grade);

    printf("%s %c ", stu[0].name, stu[0].grade);
    printf("%s %c ", stu[1].name, stu[1].grade);

    return 0;
}

void func1(student_t *stu){
    stu->name[0] = 'S';
}

void func2(char *grade){
    *grade += 2;
}
```

A. Tan F Kim B
B. San F Kim B
C. Tan F Kim D
D. San F Kim D
E. None of the above is correct.

Q5. In the process of computing **f(30)**, how many times is **f()** called (excluding **f(30)** itself)?

```
int f(int n){
    if (n <= 1) return 1;
    else if (!(n%2)) return f(n/2);
    else if (!(n%3)) return f(n/3);
    else return f(n-1);
}
```

A. 2
B. 3
C. 4
D. 5
E. 6

Q6. How many times does the swapping step occur in the process of sorting the array {20, 15, 10, 5, 1} in ascending order, using the selection sort and bubble sort algorithms as introduced in the lecture notes, respectively? (Note: In selection sort, the swapping step occurs even if the two elements to be swapped are the same element.)

A. 2 and 10
B. 2 and 15
C. 4 and 10
D. 4 and 15
E. 5 and 15

Section B: Short Structured Questions [14 marks]

Q7. Write the output of the following program. [3 marks]

```
#include <stdio.h>

void g(int *, int *, int *);

int main(void){
    int a = 1, b = 2, c = 3;

    g(&c, &a, &b);

    printf("%d %d %d\n", a, b, c);

    return 0;
}

void g(int *a, int *b, int *c){
    *b = *c/(*a)--;
}
```

Q8. Assuming N is 8, fill in the values in the 2D array **mat** at the end of this code fragment. You may leave a cell blank if its value is 0. [4 marks]

```
int mat[N][N];
int i, j, k, lower, upper, count;

for (k = 0; k < N; k += N/4) {
  lower = k;
  upper = k + N / 4;
  count = 0;

  for (i = k; i < N/4+k; i++) {
    for (j = 0; j < lower; j++)
      mat[i][j] = 0;

    for (j = lower; j < upper; j++)
      mat[i][j] = ++count;

    for (j = upper; j < N; j++)
      mat[i][j] = 0;
  }
}
```

Q9. Consider the following **mystery** function:

```
void mystery(int n, int b) {
  if (n != 0) {
    mystery(n / b, b);
    putchar('b' + n % b));
  }
}
```

What is the output of **mystery(15, 3)**? [3 marks]

Q10. Consider the following program that attempts to compute the minimum and maximum values in an integer array:

```
 1 #include <stdio.h>
 2
 3 int minMax(int [], int, int *);
 4
 5 int main() {
 6   int min, max = 0;
 7   int arr[] = {3, 2, 1, 4, 5};
 8
 9   min = minMax(arr, 5, &max);
10
11   printf("Minimum is ( %d ), ", min);
12   printf("and maximum is ( %d ).\n", max);
13
14   return 0;
15 }
16
17 int minMax(int arr[], int size, int *max) {
18   int i, min;
19
20   min = arr[0];
21   max = &arr[0];
22
23   for (i = 1; i < size ; i++) {
24     if (arr[i] > *max)
25       max = &arr[i];
26     else if (arr[i] < min)
27       min = arr[i];
28   }
29
30   return min;
31 }
```

Does this program print the correct minimum and maximum values? If not, how to fix it?

You are only allowed to change the body of the **minMax** function (*i.e.*, lines 18 to 30).

[4 marks]

Section C: Problem Solving [54 marks]

Q11. Duplicate Identification [24 marks]

You have been hired as a data scientist in a small company.

The company keeps the following information of its customers: name (a string of at most 50 characters), gender (a single character 'M' or 'F'), and age (an integer). Such information is stored in variables of a structure type called **cust_t** as given below:

```
typedef struct {
  char name[51];
  char gender;
  int age;
} cust_t;
```

Since the customer information is gathered from various sources, it is possible that some of the records are duplicates.

For example, it is possible for these two customer records: *TanAiHee F 43* and *TanAiHwa F 43*, to refer to the same person since they share very similar names, the same age and the same gender.

Your job is to identify such duplicates and sort them by similarity. Following the incremental coding technique, you are to write the following functions one by one.

(a) Write a function

```
int readCustomers(cust_t customers[])
```

that reads in the customer information from a file called **customer.dat**, stores them in an **cust_t** array called **customers**, and returns the number of customers read.

A sample of **customer.dat** is as shown on the right. There are 5 customers in the file: The first customer is called TanAiHee, female, 43 years old. The second customer is called TanAhHwee, female, 42 years old...

You should check that the file can be opened properly and handle the error when the opening of the file fails.

```
5
TanAiHee F 43
TanAhHwee F 42
TanAiHwa F 43
TanAhHwey M 43
TANAhHwah F 43
```

You may assume that the names of the customers only consist of English letters.

[5 marks]

Q11. *(continue...)*

(b) Write a function

```
float computeSimilarity(char name1[], char name2[])
```

that computes the similarity score of two given names.

Two letters in the given names are considered matched if they are the same letters (ignoring case differences) at the same position.

The similarity score of the given names is computed as the number of matched letters divided by the length of the longer name.

For example, if the given names are "TanAiHwa" and "TANAhHwah", the function returns 7/9 since there are 7 matched letters (*i.e.*, the 1st to the 4th letters, as well as the 6th to the 8th letters in the given names), and the length of the longer name (*i.e.*, the second name) is 9.

Your function should take in two names, both of which are strings stored in the char arrays **name1** and **name2**, respectively. You may assume that there is at least one letter in both names. [6 marks]

(c) Write a function

```
int findDuplicates(cust_t customers[], int size,
                   int index, cust_t duplicates[])
```

that finds all possible duplicates for a particular customer given a list of customers.

A customer record is said to be a possible duplicate of another if 1) their names have a similarity score above **0.7** based on the function in part (b), and 2) they have the same gender and age.

For example, suppose the following customer details are stored in a **cust_t** array called **customers**:

```
TanAiHee F 43
TanAhHwee F 42
TanAiHwa F 43
TanAhHwey M 43
TANAhHwah F 43
```

There are two duplicates for *TanAiHwa F 43* (*i.e.*, **customers[2]**): *TanAiHee F 43* (similarity: 6/8 ≈ 0.75) and *TANAhHwah F 43* (similarity: 7/9 ≈ 0.78).

Your function should take in a **cust_t** array called **customers**, which contains the customer information, an integer **size**, which represents the number of customers, an **index**, which represents the index of the customer whose duplicates are to be found, and another **cust_t** array called **duplicates**, for storing the possible duplicates.

Your function should return an integer which is the number of duplicates found. It should also return the possible duplicates through the **duplicates** array. [8 marks]

(d) Write a function

```
void sortDuplicates(cust_t customer,
                    cust_t duplicates[], int size)
```

that sorts a list of possible duplicates of a given customer by their similarity scores with the given customer **in descending order**. If there are multiple duplicates sharing the same similarity score with the given customer, the order among these duplicates does not matter.

For example, for the customer *TanAiHwa F 43*, after sorting the duplicates, *TANAhHwah F 43* (similarity: 7/9 ≈ 0.78) should be placed before *TanAiHee F 43* (similarity: 6/8 ≈ 0.75).

Your function should take in a **cust_t** variable called **customer**, which represents a customer, a **cust_t** array called **duplicates**, which contains the duplicates of **customer**, and an integer **size**, which is the number of duplicates in the array.

[5 marks]

Q12. Pluto English **[8 marks]**

Our spaceship just landed on the planet Pluto and we met the aliens there! Strangely, they actually do speak some language similar to English but with two major differences:

The first difference is that they add either a "wa" or "fa" after each word they speak. These two words are added in an alternating manner starting with "wa" (*i.e.*, ... wa ... fa ... wa ... fa ...).

The second difference is that they change the first occurrence of 'a' (if any) in a word to 'o'. Note that this change is not applicable to the "wa" and "fa" added as described in the first difference.

For example, if we say "i have a banana", the aliens on Pluto will say the equivalent sentence as "i *wa* h**o**ve *fa* **o** *wa* b**o**nana *fa*". (The added words are in *italic* and the changed letters are in **bold**.)

Write a function

```
void translate(char sentence[], char output[])
```

that translates a sentence from Earth English to Pluto English.

Your function should take in a char array **sentence** which contains a sentence. It translates the sentence from Human English to Pluto English, and stores the result as a string in the char array **output**.

You may assume that 1) the given sentence only contains lowercase English letters, 2) each word is separated by exactly one space, and 3) there is no space before the first word or after the last word. [8 marks]

Q13. Maximum Value Propagation **[10 marks]**

SuperSequential International Corporation owns several pipelines of data connectors where each connector in the pipeline has a capacity that can vary as wish. One such short pipeline is represented as an array as shown below, where each array element represents the capacity of a connector (in multiples of 100 Mbytes), and it is understood that data flows from left to right in the array.

{7, 3, 8, 5, 7, 10, 9, 12, 6, 5}

(a) The management would like to be able to alter the capacities of the connectors in a specific segment of a pipeline. For instance, they would like to do minimum update to the capacities, but ensure that every connector in the segment has an updated capacity that is **at least the same as** the capacity of the connector **immediately before** it.

To do that, they call a function **maxPush** that takes in an array **A** of size **N** representing a pipeline of connector capacities, as well as the starting index **i** (0 <= **i** <= **N**) and a length **d** (**d** >= 1) of a segment of the pipeline.

The following table shows the effect of **maxPush** on the array

A = {7, 3, 8, 5, 7, 10, 9, 12, 6, 5}

with **N** = 10. The **affected** segment of the pipeline is underlined for clarity.

i	d	maxPush(A, N, i d)	Remarks
3	7	{7, 3, 8, 5, 7, 10, 10, 12, 12, 12}	**A[6]**, **A[8]** and **A[9]** are updated to be equal to the final values of **A[5]**, **A[7]** and **A[8]** respectively.
2	4	{7, 3, 8, 8, 8, 10, 9, 12, 6, 5}	The value of **A[2]** (*i.e.*, 8) is propagated to the right until it meets **A[5]**, whose value is already more than 8.
6	9	{7, 3, 8, 5, 7, 10, 9, 12, 12, 12}	It is alright to specify a length that goes beyond the array bound.
3	3	{7, 3, 8, 5, 7, 10, 9, 12, 6, 5}	No change to **A** since **A[4]** is already greater than **A[3]**, and **A[5]** greater than **A[4]**.
1	1	{7, 3, 8, 5, 7, 10, 9, 12, 6, 5}	No change to **A** if the length is 1.

Write a **recursive** function

```
void maxPush(int arr[], int size, int index, int length)
```

that takes in an array **arr**, the **size** of the array, a starting **index** and a **length**. It then updates the array as described above.

You should indicate the precondition for this function.

No marks will be awarded if the function is NOT recursive. [5 marks]

Q13. *(continue...)*

(b) There is yet another update operation that the management would like to perform to alter the capacities of the connector. Specifically, they would like to do minimum update, but ensuring that every connector in the segment has an updated capacity that is **at least the same as** the capacity of the connector **immediately after** it. This is performed by a similar function called **maxPull**.

The following table shows the effect of **maxPull** on the array

$$\mathbf{A} = \{11, 3, 5, 8, 19, 7, 10, 12, 6, 9\}$$

with **N** = 10. The **affected** segment of the pipeline is underlined for clarity.

i	d	maxPull(A, N, i d)	Remarks
0	4	{11, 8, 8, 8, 19, 7, 10, 12, 6, 9}	**A[1]** and **A[2]** are updated to have the final value of **A[3]** (*i.e.*, 8), but not **A[0]**, whose value is already more than 8.
3	5	{11, 3, 5, 19, 19, 12, 12, 12, 6, 9}	Value of **A[7]** (*i.e.*, 12) is used to update **A[6]** and **A[5]**, and that of **A[4]** (*i.e.*, 19) is used to update **A[3]**.
6	9	{11, 3, 5, 8, 19, 7, 12, 12, 9, 9}	It is alright to specify a length that goes beyond the array bound.
1	1	{11, 3, 5, 8, 19, 7, 10, 12, 6, 9}	No change to the array if the length is 1.

Write a **recursive** function

```
int maxPull(int arr[], int size, int index, int length)
```

that takes in an array **arr**, the **size** of the array, a starting **index** and a **length**. It then updates the array as described above. You are to decide what value it returns.

You should indicate the precondition for this function.

No marks will be awarded if the function is NOT recursive. [5 marks]

Q14. Determinant **[12 marks]**

In linear algebra, the determinant of a square matrix is a useful value that plays an important role in finding the inverse of a matrix, among other uses. The determinant of a square matrix **A** is denoted by **det A**, or **|A|**, and can be computed from the elements of the matrix.

In the case of a 2 × 2 matrix A, the determinant is the product of the diagonal elements, subtracted by the product of the anti-diagonal elements, as shown below:

$$|\,A\,| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

For a 3 × 3 matrix B, its determinant |B| is as shown below:

$$|\,B\,| = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$= a(ei - fh) - b(di - fg) + c(dh - eg)$$

It looks complicated but there is a general pattern. To work out the determinant of a 3 x 3 matrix, we compute the following 3 terms one by one:

The 1st term is the result of multiply **a** by the determinant of the 2 x 2 matrix that is not **a**'s row or column (see below). This determinant of a smaller matrix is called a minor of the matrix. We do the same to compute the 2nd term (with **b**) and the 3rd term (with **c**). Note that the 2nd term has a negative sign. The determinant of the matrix is the sum of these 3 terms.

$$\begin{bmatrix} a & & \\ & e & f \\ & h & i \end{bmatrix} - \begin{bmatrix} & b & \\ d & & f \\ g & & i \end{bmatrix} + \begin{bmatrix} & & c \\ d & e & \\ g & h & \end{bmatrix}$$

The same procedure can be used to find the determinant of a 4 × 4 matrix (4 terms), a 5 × 5 matrix (5 terms), and so on. Note that the alternate terms (*i.e.*, the 2nd, 4th, 6th, ..., terms) have a negative sign.

Write a **recursive** function

```
int determinant(int mtx[][10], int size)
```

that computes the determinant of a square matrix.

Your function should take in a square matrix **mtx** and the **size** of the matrix. It then calculates and returns the determinant of **mtx**.

You may assume that **size** is an integer in [2, 10].

No marks will be awarded if the function is NOT recursive.

(Hint 1: The base case is when the size of the matrix is 2.)

(Hint 2: A minor of the matrix can be computed using the **determinant** function if the values of the smaller matrix are stored in a 2D array). [12 marks]

=== END OF PAPER ===